Содержание

1. Архитектура учебной ЭВМ fN8

1.1. Основные характеристики модели

– архитектура фон Неймана (общее ОЗУ для программы и данных);
– формат ячеек – 1 байт;
– объём ОЗУ – 1024 байта;
– стек – в ОЗУ;
– число РОН – 8;
– адресное пространство ввода/вывода – 2^7 адресов;
– число векторов прерываний – 8.

Структурная схема fN8 показана на рис. 1.

1.2. Организация адресного пространства

В модели fN8 реализовано три адресных пространства:

- ОЗУ – 1024 байта;
- РОН – 8 байт;
- РВУ – 128 байт.

При проектировании форматов команд стремились обеспечить для всех команд одинаковую длину – 16 бит. В форматах команд ввода/вывода предусмотрено 7-битовое поле адреса, а в регистровых командах – 3-битовое (см. форматы 10, 11, 12 и 5 на рис. 2). Это позволяет адресовать в командах весь объём пространств ввода/вывода и регистров. Однако в форматах команд, адресующих ячейки ОЗУ (6, 7, 8, 9) поле адреса составляет всего 8 бит. Разрядности счётчика команд PCL и указателя стека SPL так же составляют 8 бит. Следовательно, в этом случае поле адреса таких команд и содержимое регистров PCL и SPL определяет только *смещение в соответствующем сегменте* пространства ОЗУ; очевидно, размер сегмента составляет 256 байт.

В адресном пространстве ОЗУ размещается четыре сегмента с адресами:

Сегмент 0 – 0x000 – 0x0FF
Сегмент 1 – 0x100 – 0x1FF
Сегмент 2 – 0x200 – 0x2FF
Сегмент 3 – 0x300 – 0x3FF

причём среди этих сегментов необходимо выделить сегменты кода, данных и стека.

Для управления размещением сегментов в процессоре предусмотрен специальный программно-доступный регистр SR, в котором SR[1:0] определяет номер сегмента кода (CS), SR[3:2] – номер сегмента данных (DS), SR[5:4] – номер сегмента стека (SS). По умолчанию CS = 00, DS = 10 и SS = 11, то есть под программу отводится сегменты 0 и 1, под данные – сегмент 2, а под стек – сегмент 3.

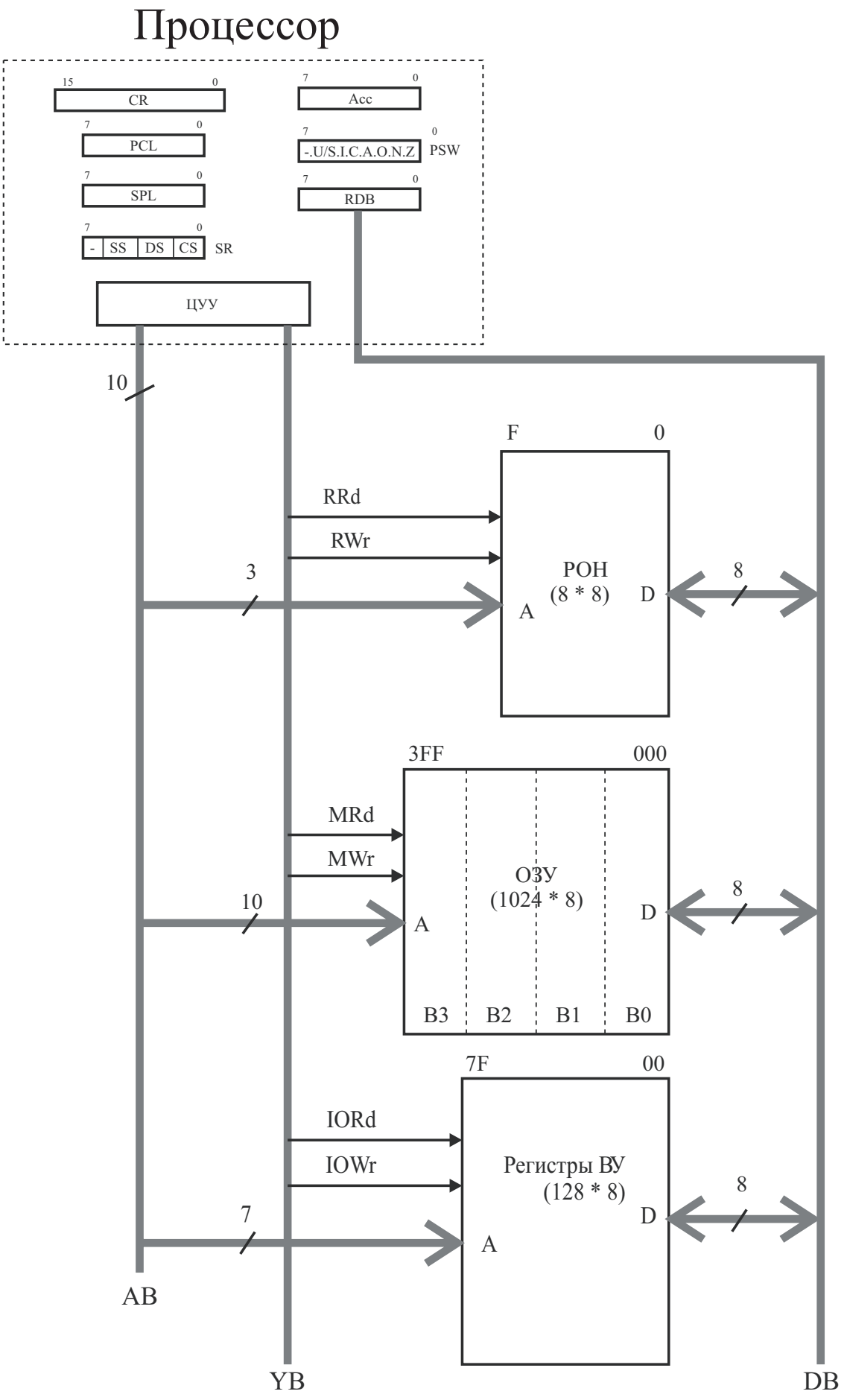

Рисунок 1. Структурная схема fN8

Таблица векторов прерываний размещается в сегменте 0 по адресам 0x00 .. 0x0F и может содержать до восьми векторов, причём *вектор 0 зарезервирован за RESET*. Каждый вектор занимает два байта и содержит 10-разрядный адрес обработчика прерывания. Значение вектора 0 по умолчанию (точка старта программы) – 0x0010.

Адресное пространство ввода/вывода составляет 128 байт, в нём может располагаться до 128 регистров внешних устройств[1] (ВУ). Разделение адресного пространства между подключаемыми ВУ осуществляется путём присвоения каждому устройству базового адреса, кратного 16. Таким образом, одновременно к системе можно подключить до 8 ВУ, каждое из которых может использовать до 16 адресов пространства ввода/вывода.

1.3. Директивы компилятора

При написании программы на языке Ассемблер можно пользоваться следующими директивами компилятора:

.c <сегмент> – выбирает текущий сегмент компиляции;

.org <адрес> – изменяет текущий адрес компиляции;

.db <байт, байт, ... байт> – загрузка констант размером в 1 байт в текущий сегмент начиная с текущего адреса компиляции;

.dw <слово, метка, ... слово> – загрузка констант размером в 2 байта в текущий сегмент начиная с текущего адреса компиляции.

В процессе компиляции возможна загрузка констант в сегмент данных (например, таблицы ASCII-кодов символов). По умолчанию сегменты «0» и «1» отведены для кода, сегмент «2» – для данных, а сегмент «3» – для стека. Если (лучше в конце текста программы) поставить директиву **.c 2**, то компилятор продолжит компиляцию фактически в сегмент данных. Например, если требуется разместить таблицу 7-сегментных кодов[2] десятичных цифр в сегменте данных начиная с адреса 0x90, то в конце текста программы можно добавить такой фрагмент:

$.c\ 2$
$.org \quad 0x90$
$.db \quad 0x3F,\ 0x06,\ 0x5B,\ 0x66,\ 0x6D,\ 0x7D,\ 0x07,\ 0x7F,\ 0x6F,\ 0x77$

Использование этих директив позволяет изменить установленную по умолчанию точку старта программы – 0x10, например, на адрес 0x20 (a). Ещё проще поставить в нужном месте метку, например, *Start:* и объявить её точкой старта (b).

a)	*b)*
$.org \quad 0$	$.org \quad 0 \qquad .$
$.db \quad 0x20, \quad 0x0$	$.dw \quad Start$

Эти же директивы позволяют автоматизировать заполнение **таблицы векторов прерываний**. Таблица векторов прерываний в fN8 размещается в младших адресах памяти 0x000 – 0x00F, каждый вектор занимает два байта. Вектор 0 определяет точку

[1] При соответствующей организации схемы ВУ допускается присваивание одинакового адреса двум различным регистрам, из которых один является регистром ввода, а другой – регистром вывода

[2] В этом примере сегменту **A** индикатора соответствует младший бит кода.

старта (по умолчанию — 0x010), вектора 1, 2, 3 и 4 по умолчанию присваиваются клавиатуре, таймеру-2, таймеру-5 и контроллеру 7-сегментной индикации соответственно. Остальные вектора (5 — 7) можно использовать для других разрабатываемых внешних устройств или для подключения нескольких экземпляров одинаковых ВУ с разными векторами. Адрес младшего байта вектора определяется как его удвоенный номер.

Если поставить метки в начале всех обработчиков прерываний, то загрузка векторов прерываний может выглядеть следующим образом (для случая, когда вектора определяются последовательно):

.org 2
.dw IntKey, IntTim2, IntTim5, Int7Seg

По адресам неиспользуемых векторов в таблице векторов прерываний сохраняются значения 0. Если в системе используются те ВУ, вектора прерываний которых по умолчанию следуют не подряд, то можно или при подключении поменять им вектора в желаемом порядке или в директиве заполнения таблицы по неиспользуемым адресам записывать 0, например:

.org 2
.dw IntKey, 0, 0, Int7Seg ,

если подключать только контроллер клавиатуры и контроллер 7-сегментной индикации и оставить им назначаемые по умолчанию вектора.

1.4. Ресурсы процессора

Вне адресных пространств – в процессоре – расположены программно-доступные объекты:

- Acc[7:0] – аккумулятор;
- PCL[7:0] – счётчик команд;
- SPL[7:0] – указатель стека;
- SR[7:0] – сегментный регистр;
- PSW[7:0] – регистр слова состояния процессора;

Регистр слова состояния процессора включает следующие флаги:

PSW[0] = Z – нулевой результат;
PSW[1] = N – отрицательный результат;
PSW[2] = O – арифметическое переполнение;
PSW[3] = AC – дополнительный перенос;
PSW[4] = C – перенос;
PSW[5] = I – разрешение прерывания;

Объекты процессора CR[15:0] – регистр команд и DR[7:0] – регистр данных не являются программно-доступными и отображают код выполняемой команды и значение второго операнда бинарной операции АЛУ соответственно.

1.5. Подсистема прерываний fN8

В модели fN8 предусмотрен механизм векторных внешних прерываний. Внешние устройства формируют запросы на прерывания, которые поступают на входы *контроллера прерываний*. При подключении ВУ, способного формировать запрос на прерывание, ему ставится в соответствие номер входа контроллера прерываний — вектор прерывания, принимающий значение в диапазоне [1..7].

Контроллер передает вектор, соответствующий запросу, процессору, который начинает процедуру обслуживания прерывания.

Каждому из контролируемых системой прерываний событий должен соответствовать т. н. *обработчик прерывания* — подпрограмма, вызываемая при возникновении события конкретного прерывания.

Механизм прерываний, реализованный в fN8, поддерживает *таблицу векторов прерываний*, которая создается в оперативной памяти (1.3), может содержать до 8 векторов и располагается по адресам 00 .. 0F нулевого сегмента. Каждый вектор занимает 2 байта памяти и содержит 10-разрядный адрес начала обработчика соответствующего события. Вектор 0 (ячейки 00 и 01) зарезервирован за точкой старта программы по кнопке *Запуск*.

Процессор начинает обработку прерывания[3], завершив текущую команду. При этом он

1) получает от контроллера вектор прерывания;
2) формирует и помещает в верхушку стека двухбайтовое слово, 10 младших разрядов которого – адрес возврата (текущее состояние CS.PCL), а 6 старших – флаги PSW[5:0];
3) сбрасывает в «0» флаг разрешения прерывания I;
4) извлекает из таблицы векторов прерываний адрес обработчика, соответствующий обслуживаемому вектору, и помещает его в CS.PCL, осуществляя тем самым переход на подпрограмму обработчика прерывания.

Таким образом, вызов обработчика прерывания, в отличие от вызова подпрограммы связан с помещением в стек не только адреса возврата, но и текущего значения вектора флагов. Поэтому последней командой подпрограммы обработчика должна быть команда IRET, которая не только возвращает в PC адрес команды, перед выполнением которой произошло прерывание но и восстанавливает те значения флагов, которые были в момент перехода на обработчик прерывания.

Не всякое событие, которое может вызвать прерывание, приводит к прерыванию текущей программы.

В состав процессора входит программно-доступный флаг I разрешения прерывания. При I = 0 процессор не реагирует на запросы прерываний. После сброса процессора флаг I так же сброшен и все прерывания запрещены. Для того, чтобы разрешить прерывания, следует в программе выполнить команду EI `(Enable Interupt)`.

Выше отмечалось, что при переходе на обработчик прерывания флаг I автоматически сбрасывается, в этом случае прервать обслуживание одного прерывания другим прерыванием нельзя. По команде IRET значение флагов восстанавливается, в том числе

[3]Если прерывания разрешены

вновь устанавливается $I = 1$, следовательно в основной программе прерывания опять разрешены.

Если требуется разрешить прерывания (другие!) в обработчике прерывания, достаточно в нём выполнить команду EI. Контроллер прерываний и процессор на аппаратном уровне блокируют попытки запустить прерывание, если его обработчик начал, но не завершил работу.

Таким образом, флаг I разрешает или запрещает все прерывания системы. Если требуется выборочно разрешить некоторое подмножество прерываний, используются программно-доступные флаги разрешения формирования запросов прерываний непосредственно на внешних устройствах.

Как правило каждое ВУ, которое может вызвать прерывания, содержит в составе своих регистров управления разряды флагов разрешения прерываний, по умолчанию установленных в «0». Если оставить некоторые (или все) эти флаги в нуле, то внешнему устройству будет запрещено формировать соответствующие запросы контроллеру прерываний.

Иногда бывает удобно (например, в режиме отладки) иметь возможность вызвать обработчик прерывания непосредственно из программы. Поэтому в системах команд многих ЭВМ, в том числе и fN8, имеются команды вызова прерываний — INT n, где n — вектор прерывания. Процессор, выполняя команду INT n, производит те же действия, что и при обработке прерывания с вектором n.

Характерно, что с помощью команды INT n можно вызвать обработчик прерывания даже в том случае, когда флаг разрешения прерывания сброшен.

2. Система команд

2.1. Система операций и форматы

Система команд включает в себя следующие операции: арифметические и логические операции над аккумулятором (Acc) и ячейкой памяти или регистром с размещением результата в Acc, команды управления битами, команды пересылки, ввода/вывода, передачи управления (включая вызовы подпрограмм), управление прерываниями.

Адресация в командах с ячейками ОЗУ – прямая и непосредственная. В регистровых командах – прямая, косвенная и несколько вариантов автоиндексной адресации. Адресация в командах ввода/вывода – только прямая. Кроме того, возможна адресация отдельных битов в любой ячейке сегмента данных ОЗУ или в любом РВУ.

Все команды имеют размер 16 бит. Большинство команд при этом являются одноадресными.

Форматы всех команд fN8 показаны на рис. 2

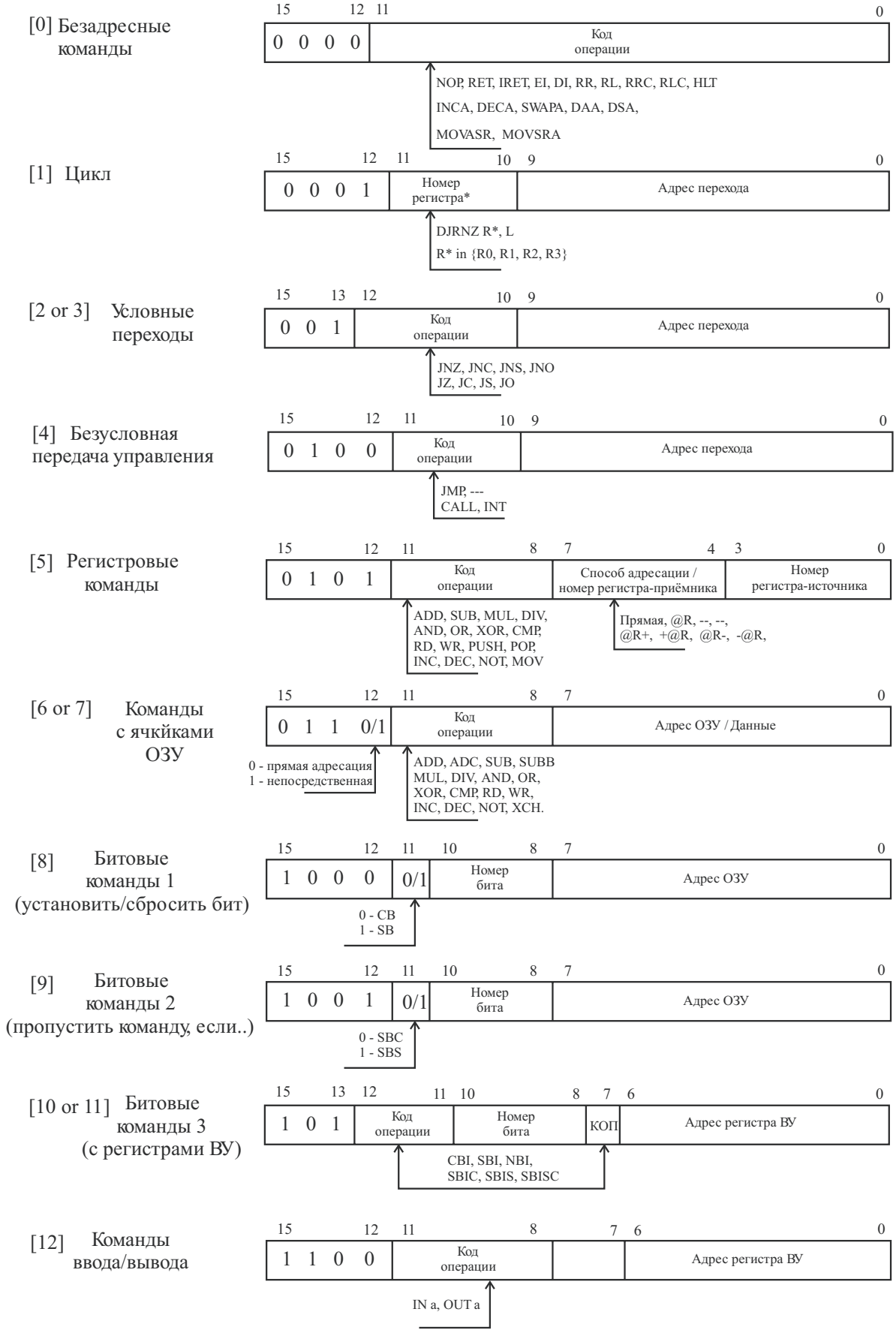

Рисунок 2. Форматы команд fN8

2.2. Таблица команд

В таблице приняты следующие обозначения:

Acc – содержимое аккумулятора;

DD – содержимое ячейки памяти или непосредственный операнд;

R – содержимое регистра общего назначения R;

R* – содержимое регистра или косвенно адресуемой через регистр ячейки памяти;

p – префикс перед именем регистра, определяющий способ адресации;

A – адрес ячейки памяти данных;

M(A) – содержимое ячейки памяти данных по адресу A;

C – флаг PSW[4] переноса(заёма)

SPL – содержимое указателя стека;

L – адрес перехода (метка или абсолютный);

CR – содержимое регистра команд;

RIO - содержимое регистра внешнего устройства;

a - номер (адрес) регистра внешнего устройства;

b - номер бита в байте;

Таблица 1 — Система команд

Мнемокод	Название	Действия	Флаги O C A N Z
Безадресные команды [Код 0]			
NOP	Нет операции	PCL := PCL + 1	- - - - -
RET	Возврат из подпрограммы	PCL := M(SS.SPL); Inc(SPL); CS := M(SS.SPL)[1:0]; Inc(SPL)	- - - - -
IRET	Возврат из прерывания	PCL := M(SS.SPL); Inc(SPL); PSW[5:0].CS := M(SS.SPL); Inc(SPL)	- - - - -
EI	Разрешить прерывание	FI := 1	- - - - -
DI	Запретить прерывание	FI := 0	- - - - -
RR	Сдвиг аккумулятора правый циклический	Acc[7:0] := Acc[0].Acc[7:1]; FC := Acc[0]	- - - - -
RL	Сдвиг аккумулятора левый циклический	Acc[7:0] := Acc[6:0].Acc[7]; FC := Acc[7]	- - - - -
RRC	Сдвиг аккумулятора правый через перенос	Acc[7:0] := FC.Acc[7:1]; FC := Acc[0]	- - - - -
RLC	Сдвиг аккумулятора левый через перенос	Acc[7:0] := Acc[6:0].FC; FC := Acc[7]	- - - - -
Смотри продолжение на следующей странице			

Таблица 1 — Система команд

Мнемокод	Название	Действия	Флаги O	C	A	N	Z
NOTA	Инверсия аккумулятора	$Acc := \overline{Acc}$	0	0	0	+	+
INCA	Инкремент аккумулятора	Acc := Acc + 1	-	+	+	+	+
DECA	Декремент аккумулятора	Acc := Acc – 1	-	+	+	+	+
SWAPA	Обмен тетрадами аккумулятора	Acc[7:4] ⇔ Acc[3:0]	-	-	-	-	-
DAA	Десятичная коррекция сложения	См. раздел 2.3.1	-	+	+	+	+
DSA	Десятичная коррекция вычитания	См. раздел 2.3.1	-	+	+	+	+
MOVSP	Загрузка SPL	SPL := Acc;	-	-	-	-	-
MOVAPSW	Прочитать PSW	Acc := PSW	-	-	-	-	-
MOVASR	Прочитать SR	Acc := SR	-	-	-	-	-
MOVSRA	Загрузить SR	SR := Acc	-	-	-	-	-
HLT	Стоп	Прекратить командные циклы	-	-	-	-	-
Цикл [Код 1]							
DJRNZ R^c, L	Цикл	$R^c := R^c - 1$; if $R^c \neq 0$ then goto L	-	-	-	-	-
$R^c \in \{R0, R1, R2, R3\}$							
Команды условных переходов [Код 2 or 3]							
JNZ L	Переход, если не ноль	if Z=0 then CS.PCL := CR[9:0]	-	-	-	-	-
JNC L	Переход, если не перенос	if C=0 then CS.PCL := CR[9:0]	-	-	-	-	-
JNN L	Переход, если не отрицательно	if N=0 then CS.PCL := CR[9:0]	-	-	-	-	-
JNO L	Переход, если не переполнение	if O=0 then CS.PCL := CR[9:0]	-	-	-	-	-
JZ L	Переход, если ноль	if Z=1 then CS.PCL := CR[9:0]	-	-	-	-	-
JC L	Переход, если перенос	if C=1 then CS.PCL := CR[9:0]	-	-	-	-	-
JN L	Переход, если отрицательно	if N=1 then CS.PCL := CR[9:0]	-	-	-	-	-
JO L	Переход, если переполнение	if O=1 then CS.PCL := CR[9:0]	-	-	-	-	-
Смотри продолжение на следующей странице							

Таблица 1 — Система команд

Мнемокод	Название	Действия	Флаги O	C	A	N	Z
Команды безусловной передачи управления [Код 4]							
JMP L	Безусловный переход	CS.PCL := CR[9:0]	-	-	-	-	-
CALL L	Вызов подпрограммы	Dec(SPL); M(SS.SPL)[1:0] := CS; Dec(SPL); M(SS.SPL) := PCL; CS.PCL := CR[9:0]	-	-	-	-	-
INT v	Вызов прерывания	Dec(SPL); M(SS.SPL) := PSW[5:0].CS; Dec(SPL); M(SS.SPL) := PCL; PCL := M(2v); CS := M(2v+1)[1:0]; FI := 0	-	-	-	-	-
v = CR[2:0] – вектор прерывания							
Регистровые команды [Код 5] (Для команд ADC и SUBB – [Код F])							
ADD pR	Сложение	Acc := Acc + R*	+	+	+	+	+
ADC pR	Сложение с переносом	Acc := Acc + R* + C	+	+	+	+	+
SUB pR	Вычитание	Acc := Acc – R*	+	+	+	+	+
SUBB pR	Вычитание с заёмом	Acc := Acc – R* – C	+	+	+	+	+
MUL pR	Умножение	R7.Acc := Acc × R*	-	-	-	-	-
DIV pR	Деление	Acc := Acc : R*	-	-	-	-	+
AND pR	Конъюнкция	Acc := Acc & R*	0	0	0	+	+
OR pR	Дизъюнкция	Acc := Acc ∨ R*	0	0	0	+	+
XOR pR	Неравнозначность	Acc := Acc ⊕ R*	0	0	0	+	+
CMP pR	Сравнение	R* – Acc	+	+	+	+	+
RD pR	Чтение	Acc := R*	-	-	-	-	-
WR pR	Запись	R* := Acc	-	-	-	-	-
XCH pR	Обмен	R* ⇔ Acc	-	-	-	-	-
PUSH R	Поместить в стек	Dec(SPL); M(SPL) := R;	-	-	-	-	-
POP R	Извлечь из стека	R := M(SPL); Inc(SPL)	-	-	-	-	-
INC R	Инкремент	R := R + 1;	-	+	+	+	+
DEC R	Декремент	R := R – 1;	-	+	+	+	+
NOT R	Инверсия	$R := \overline{R}$;	0	0	0	+	+
MOV R_r, R_t	Копирование	$R_r := R_t$	-	-	-	-	-
Смотри продолжение на следующей странице							

Таблица 1 — Система команд

Мнемокод	Название	Действия	Флаги O	C	A	N	Z
Команды с ячейками ОЗУ [Код 6 or 7] Код 6 – прямая адресация (A), 7 – непосредственная (#A)							
ADD A	Сложение	Acc := Acc + DD	+	+	+	+	+
ADC A	Сложение с переносом	Acc := Acc + DD + FC	+	+	+	+	+
SUB A	Вычитание	Acc := Acc – DD	+	+	+	+	+
SUBB A	Вычитание с заёмом	Acc := Acc – DD – FC	+	+	+	+	+
MUL A	Умножение	R7.Acc := Acc × DD	-	-	-	-	-
DIV A	Деление	Acc := Acc : DD	-	-	-	-	+
AND A	Конъюнкция	Acc := Acc & DD	0	0	0	+	+
OR A	Дизъюнкция	Acc := Acc ∨ DD	0	0	0	+	+
XOR A	Неравнозначность	Acc := Acc ⊕ DD	0	0	0	+	+
CMP A	Сравнение	DD – Acc	+	+	+	+	+
RD A	Чтение	Acc := DD	-	-	-	-	-
WR A	Запись	M(A) := Acc	-	-	-	-	-
XCH A	Обмен	M(A) ⇔ Acc	-	-	-	-	-
INC A	Инкремент	M(A) := M(A) + 1	-	+	+	+	+
DEC A	Декремент	M(A) := M(A) – 1	-	+	+	+	+
NOT A	Инверсия	M(A) := $\overline{DD}$	0	0	0	+	+
Битовые команды 1 [Код 8]							
CB A,b	Сбросить бит	M(A)[b] := 0	-	-	-	-	-
SB A,b	Установить бит	M(A)[b] := 1	-	-	-	-	-
Битовые команды 2 [Код 9]							
SBC A,b	Пропустить следующую команду, если бит сброшен	if M(A)[b]=0 then PCL := PCL + 1	-	-	-	-	-
SBS A,b	Пропустить следующую команду, если бит установлен	if M(A)[b]=1 then PCL := PCL + 1	-	-	-	-	-
Битовые команды 3 [Код A or B]							
CBI a,b	Сбросить бит	RIO(a)[b] := 0	-	-	-	-	-
SBI a,b	Установить бит	RIO(a)[b] := 1	-	-	-	-	-
NBI a,b	Инвертировать бит	$RIO(a)[b] := \overline{RIO(a)[b]}$	-	-	-	-	-
SBIC a,b	Пропустить следующую команду, если бит сброшен	if RIO(a)[b]=0 then PCL := PCL + 1	-	-	-	-	-

Смотри продолжение на следующей странице

Таблица 1 — Система команд

Мнемокод	Название	Действия	Флаги O	C	A	N	Z
SBIS a,b	Пропустить следующую команду, если бит установлен	if RIO(a)[b]=1 then PCL := PCL + 1	-	-	-	-	-
SBISC a,b	Пропустить следующую команду, если бит установлен и сбросит бит	if RIO(a)[b]=1 then (PCL := PCL + 1; RIO(a)[b] := 0)	-	-	-	-	-
Команды ввода/вывода [Код C]							
IN a	Ввод	Acc := RIO(a)	-	-	-	-	-
OUT a	Вывод	RIO(a) := Acc	-	-	-	-	-

2.3. Пояснения к некоторым командам таблицы 1

2.3.1. Безадресные команды десятичной коррекции

DAA – десятичная коррекция байта в Acc после сложении (кодировка «8421»)

$$\text{if } (B_L > 9) \text{ or } (AC) \text{ then } B := B + 0x06; \tag{1}$$

$$\text{if } (B_H > 9) \text{ or } (C) \text{ then } B := B + 0x60; \tag{2}$$

DSA – десятичная коррекция байта в Acc после вычитании (кодировка «8421»)

$$\text{if } (AC) \text{ then } B := B - 0x06; \tag{3}$$

$$\text{if } (C) \text{ then } B := B - 0x60; \tag{4}$$

Здесь B – предварительный результат операции сложения (вычитания), B_H – старшая тетрада B, B_L – младшая тетрада B, AC – перенос (заём) из младшей тетрады, C – перенос (заём) из байта.

2.3.2. Умножение

Сомножители рассматриваются как целые беззнаковые числа. Содержимое аккумулятора можно умножить на содержимое регистра, прямо или косвенно адресуемой ячейки памяти или 8-разрядную константу. Формат произведения – два байта, поэтому младший байт размещается в аккумуляторе Acc, а старший – в регистре R7.

2.3.3. Деление

Целочисленное беззнаковое деление: байт Acc делится на байт делителя, целая часть частного помещается в Acc. Если содержимое Acc меньше делителя, то результат деления равен 0, устанавливается флаг $Z = 1$. В случае ненулевого результата деления

флаг $Z = 0$, остальные флаги в операции деления не изменяются. В качестве делителя можно использовать непосредственный операнд, содержимое регистра, содержимое прямо или косвенно адресуемой ячейки памяти.

2.3.4. Цикл

DJRNZ R,L – декремент регистра и проверка. Если после декремента содержимое регистра $\neq 0$, то осуществляется переход по указанному адресу, иначе – на следующую команду. Можно использовать для организации циклов. Работает только с регистрами R0, R1, R2, R3.

2.3.5. Команды безусловной передачи управления

CALL L – вызов подпрограммы по прямому адресу. Использует две ячейки стека для размещения 10-разрядного адреса возврата (в старшем байте занято только два младших разряда).
INT v – вызов обработчика прерывания по вектору v. Как и команда CALL, сохраняет адрес возврата, а в свободные 6 разрядов старшего байта помещает флаги I, O, C, AC, N, Z. Таблица векторов прерываний в ОЗУ начинается с адреса 0x000, вектор прерываний $v \in \{0, 1, \ldots 7\}$. Адрес обработчика прерываний загружается из ячеек с адресами (2v) и (2v+1)[1:0] в PCL и SR[1:0] соответственно.